# Semantic Information Retrieval With RNNs

Rahul Pandey
Stanford University
rkpandey@stanford.edu

Arun Prasad
Stanford University
akprasad@stanford.edu

December 16, 2011

## Abstract

We discuss the application of recursive neural networks to domains with recursive structure and describe our attempts to build a model capable of matching sentence pairs and, by extension, of finding the answer that most corresponds to some given question. Our early results are poor, but they indicate that our model is correct and can be competitive given enough time and data.

## 1 Introduction

Current IR systems are capable of answering "factoid" questions relating to specific pieces of information, but they are much less successful at answering "non-factoid" questions, which often require longer and more detailed answers. This is because although keyword searches are able to search text data based on orthographic content and word-level semantics,[1] they are unable to understand the sort of sentence-level semantics required for answering nontrivial questions.

For this reason, research into building a system capable of non-factoid question answering is ongoing. Surdeanu et al. demonstrated the ability of a linguistically-oriented classifier to provide answers to non-factoid questions on the Yahoo! Answers data set, and elements of their approach appear in several other QA systems [4]. Here, however, we describe a more generic approach using recursive neural networks (RNNs).

In order to compare our model with Surdeanu et al., we re-implemented their basic setup, as described further below. Using this setup, we obtained rather disheartening MRR scores for various training sizes, all of which are slightly above our baseline score. But we have determined several factors that impacted the quality of results, and on this basis we are confident that our model can be competitive in the future.

[1]Which are mainly limited to stemming and synonym resolution.

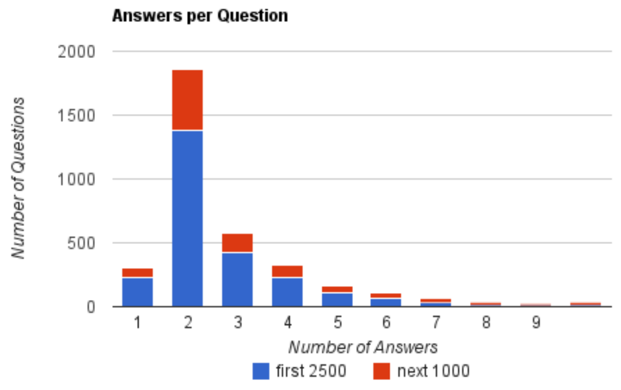
## 2 Recursive neural networks

A recursive neural network (RNN) has the topology of a standard neural network but is applied recursively to the data, which generally takes the form of a directed acyclic graph (DAG) [1]. The application of the neural network starts from the child nodes of the DAG and proceeds upward toward the parents. If our DAG happens to be a tree, we start from the leaf nodes and proceed upward toward the root, upon which we can stack some sort of classification step.

RNNs have been shown to work effectively in interpreting items with recursive structure, including English sentences and natural scene images [2]. But recursive neural networks are also useful in detecting sentence paraphrases [3], which implies that they can deal with sentences at a level of abstraction sufficient for matching some user-provided question with one among a set of answers. This, in turn, implies that they are well suited to the task of non-factoid QA.

## 3 The Yahoo! Answers data set

The Yahoo! Answers data set used by Surdeanu et al. consists of 142,627 questions and at least one answer for each. Among these answers, one is chosen as a "best" answer, as selected by users of Yahoo! Answers. Due to time constraints, we examined only a small subset of this large number of questions. Since Surdeanu et al. seem to measure the size of their test set in terms of question-answer pairs, we chose a sufficient number of sentences to produce either 5000 or 10,000 question-answer pairs. According to the statistics below, we found that 1931 questions produced 5000 pairs and 3537 questions produced 10,000 pairs.



As evidenced by our data set, most questions have 2 or 3 answers, which indicates that even a classifier that randomly picks a best answer can obtain respectable scores for MRR and P@1. This is part of the justification for following the approach used by Surdeanu et al. and using IR to obtain a larger answer set for each question.

## 4 Building the pipeline

Having elaborated on both our task and the nature of RNNs, we describe our basic pipeline below.

### 4.1 Information Retrieval

Although our model is capable of evaluating the likelihood that some question from the question set $Q$ has a particular answer in the answer set $A$, it takes long enough to do so that an examination of every item in $A$ is not feasible. Per Surdeanu et al., therefore, we proceed with classification only after using an information retrieval (IR) system to select a set of $N$ candidate answers for each question. More specifically, we used Terrier, an open-source Java IR system, to find $N$ possible answers among all answers $a \in C$, where $C$ contains only those answers that are the best answer for some $q \in Q$. In theory, $|C| = 142627$, but due to persistent issues with our starting data – including a gap of about 40,000 missing questions and the sporadic appearance of malformed parse trees – our starting set was less substantial.

Having completed this initial IR step, we consider only those questions for which Terrier retrieved the correct answer. This constraint leaves us with a final question set $Q_c$, which is about one-fifth of the size of $C$. The motivation for considering only those answers in $C$ is that they are generally grammatical and well-formed, which means that our classifier will tend to do a better job of learning from them. Many of the answers in the data set are not of very high quality (see table 1).

| | |
|---|---|
| High Quality | **Question**: How do mobile phones work?<br>**Answer**: Mobile phones connect to a cellular network provided by a network operator. They are able to send radio signals to satellites while moving across a wide geographic area, thus giving them "mobile" capabilities. |
| Low Quality | **Question**: How do I make my dog listen to me?<br>**Answer**: very carefully. |

Table 1: Examples of responses of varying quality.

Still, parts of the setup used by Surdeanu et al. are ambiguous. Our understanding is that for a given question $q$ in the training set, Surdeanu et al. trained on all of the answers corresponding to $q$, including answers in $A$. However, the testing procedure that they followed seems only to consider those questions in $C$ that were retrieved by the IR system. Later on, we will discuss the implications of this approach and describe our results for a different training procedure.

### 4.2 Forward propagation

Regardless of the nature of the IR step, our classifier still retrieves two blocks of text: a question $q$, and some answer $a$. Each block is split into sentences, which are in turn parsed using the Stanford parser. Within each tree, each node – including both leaf-level "word" nodes and parent "clause" nodes – is represented as an $n$-dimensional vector. We developed our system for $n = 10$, but we hope to use $n = 50$ in the future.

For each sentence pair $(s_q, s_a)$ drawn from $q$ and $a$, we construct a similarity matrix, which consists of the Euclidean distance between between all nodes in both sentence trees. However, since sentences can

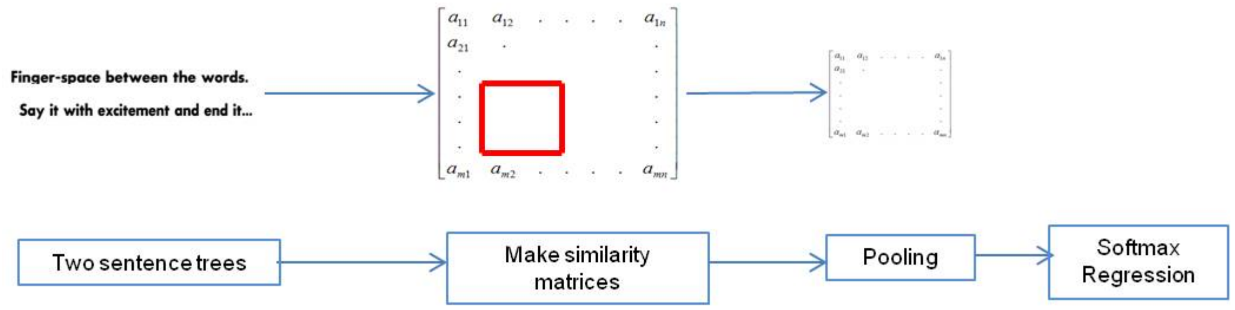

Finger-space between the words.
Say it with excitement and end it...
Two sentence trees
Make similarity matrices
Pooling
Softmax Regression

Figure 1: The forward pipeline of our model. The error from the softmax layer is used for backpropagation.

be rather complex, the similarity matrix could have thousands of elements, the result of which is that classification time increases greatly. Moreover, sentences can be of different lengths, so the dimensions of our similarity matrix could change for each $(q, a)$ pair. To minimize our feature space and train faster, we use average pooling to reduce the complexity of the similarity matrix.

### 4.3 Softmax

As a result of the previous step, we have a set of matrices $\{M_1, M_2, \cdots, M_n\}$, where $n$ is the number of possible sentence pairs. For each of these matrices, we perform softmax classification, but since we mean to perform a binary classification – we determine whether or not the similarity matrix indicates a match – this is simply logistic regression.

After reshaping a similarity matrix $M$ of dimensions $m \times n$ into a vector of length $mn$, we run softmax normally. This step generates the probability that the similarity matrix indicates a match. By calculating the norm between our analytical gradient and the numerical gradient, we verified that our implementation of softmax is correct.

### 4.4 Backpropagation

Once we have a probability score from the softmax classifier, we can compare it to the expected score (either 1 or 0) to find the scalar error term $\delta_i$ for each reshaped vector $v_i$. We then find $\nabla_{v_i}\delta$, reshape the gradient to match the dimensions of $M_i$, and backpropagate the expected error terms through the similarity matrices and to every node in the sentence trees that generated $M_i$.

The goal at the end of backpropagation is to capture semantic information. For example, the words “cat” and “feline” or the phrases “We rushed to the demonstration” and “I ran to the protest” are similar in meaning, so we desire that the corresponding vector representations in the sentence trees should be similar. Per the usual neural network procedure, the error terms that we calculate in training are used to help our weight vectors fit to the data and, by extension, capture some of the semantic information inherent in the training data.

## 5 Results and error analysis

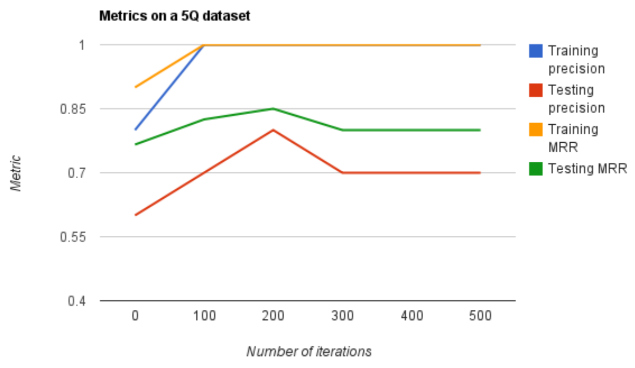
After determining the vector representation of each clause from pre-training, we train the weights in our softmax classifier and refine our vector representations through forward- and backpropagation. For testing, we iterate through a list of test questions and find the top $N$ results, as indicated by Terrier. If the correct answer is not among those results, we remove the question from consideration. Otherwise, we test the classifier to see if it correctly selects the best answer.

We tested our model with several different regularization parameters. Due to problems with using L-BFGS (described further below), we also experimented with different learning rates for stochastic gradient descent (SGD): $10^{-3}, 10^{-5}$, and $10^{-9}$. Of these three rates, only the first was successful; the others produced no noticeable changes in our weights. As for regularization parameters, we experimented with several combinations of values, but our learning rate was too small to show their effect for the number of iterations we tried.

When evaluating results, we use two metrics: precision at 1 (P@1) and mean reciprocal rank (MRR). With $N = 15$ and a baseline system that randomly guesses, we expect P@1 $= 6.67\%$ and MRR $= 30.1\%$. As indicated earlier, our results were not much better than this baseline. With SGD and $\alpha = 10^{-3}$, we have the following results:

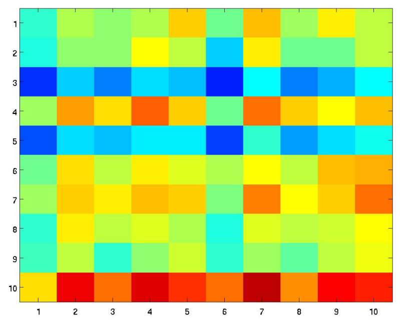
| # of questions | Best MRR | Best P@1 |
|---|---|---|
| 100 | 0.2008 | 0.0606 |
| 1931 | 0.2251 | 0.0727 |
| 2500 | 0.2165 | 0.0788 |
| 3537 | 0.1989 | 0.0545 |

As indicated by the data above, our model does not perform effectively in testing. (The data in the last row is likely an artifact of SGD.) However, we do have data that shows that our model is learning. When testing on a toy data set of 5 questions, we find that the model fits to the training data, as we would expect:

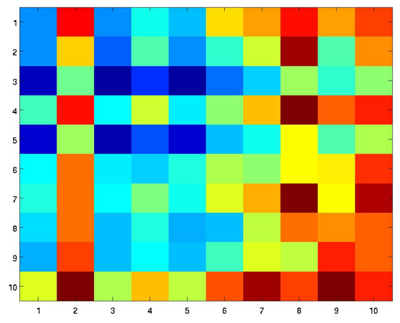


Moreover, by visualizing our pooled similarity matrices we were able to obtain some examples of sentences that indicate good similarity and sentences that indicate bad similarity. For the question `How does China view the trustworthiness of North Korea?` and the answers that follow, we had the following similarity matrices. Bluer matrices indicate better matches.

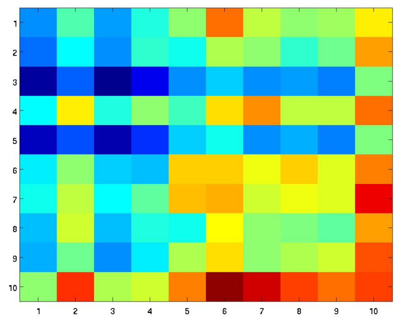
- `November 9, 1966 Indica Gallery in London during a performance art installation by Ono.`



- `The Japanese eat the most seafood, particularly raw seafood.`



- `Koreans do no differentiate between hi and how are you like we do in English.`



These sentences are arranged from less likely to more likely. Note that the region in the upper-left becomes bluer with the appearance of the word `Japanese`, which is a name for a group of people from a particular country. The same applies to the word `Koreans`, which moreover is closely aligned with the word `Korea`. Correspondingly, the region in the upper left becomes more blue. In addition, the second column, which was originally very red, became much less so. This could be due to both the appearance of `English`, which could be interpreted as an ethnonym, and the disappearance of `seafood`, which has little to do with trustworthiness or national policy.

The question, then, is why our model performs so poorly. While we continued to test our model, we came up with three avenues for further exploration.

First and most importantly, we should test the classifier on a larger data set. Surdeanu et al. made use of a training set that was more than an order of magnitude larger than the one we were compelled to employ, both because of time constraints and because of gaps and malformed trees in the data set we inherited.[2]

Second, we should adjust our model to work with a more reliable method of numerical optimization that is more able to deal with nonlinear functions, like L-BFGS. This is not to say that we did not try using L-BFGS; rather, it had no visible effect on our calculated weights. We are at a loss as to how to explain this, since the code that calls the L-BFGS package[3] to update our weights is nearly identical to the code that calls our SGD package. Had we spent more time investigating the code we inherited, we might have been able to get this method to run correctly.

Third, we should investigate a different training method. According to our understanding of Surdeanu et al., their linguistically oriented model was trained over all answers given to some sentence $q$. But although Surdeanu et al. pruned the data set to remove obviously bad answers, retaining only those answers with "at least four words each, out of which at least one is a noun and at least one is a verb," many of the answers remaining are still fairly bad, or else just superficial restatements of the original question. Based on the nature of the RNN, we hypothesize that our model is learning to discriminate between bad answers and good answers without learning to discriminate among good answers at all. For that reason, we think it worthwhile to train on some of the best answer data produced by Terrier instead. In doing so, our model might better understand those qualities that make an answer a particularly good answer, especially in the context of other similar answers. This is certainly a possible explanation for our poor performance; when examining the probabilities for certain answers over many iterations, we found that all good answers were initially ranked very lowly ($\approx 30-50\%$) and uniformly came to be ranked very highly ($\approx 60-98\%$). This indicates that our model may be learning, but perhaps in the wrong way.

Fourth, we should experiment with different pretraining methods. Most of our tests were implemented with 10-dimensional word vectors calculated via 20,600 iterations of SGD. However, we also experimented with a set of word vectors that was part of the code base before we began work, as well as the 50-dimensional word vectors created by Collobert and Weston. (These were reduced to 10-dimensional vectors via PCA.) Neither of these alternate word vector sets had an appreciable effect on the performance of the model, which indicates either that there is an undetected bug in our model or that we did not use enough training data or the right sort of numerical optimization method.

Clearly, there is much more work to be done with this particular application of RNNs, and although we were unable to complete much of it, we have a strong indication of what to explore next. Our performance in these small cases was negligible, but we remain confident that our model can be a successful part of a non-factoid QA system.

## A. Collaboration

We conducted our work within the context of the deep learning research group under the supervision of Richard Socher. We worked alongside Jean Feng (former CS229 student), Imran Thobani, Reed Nightingale, and Deepak Merugu (all currently CS229 students). We would like to acknowledge and thank them for their help.

## References

[1] Socher, R. Learning continuous phrase representations and syntactic parsing with recursive neural networks. *NIPS* (2010).

[2] Socher, R. Parsing natural scenes and natural language with recursive networks. *ICML* (2011).

[3] Socher, R., Huang, E. H., Pennington, J., Ng, A. Y., and Manning, C. D. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. *NIPS* (2011).

[4] Surdeanu, M., Ciaramita, M., and Zaragoza, H. Learning to rank answers to non-factoid questions from web collections. *Computational Linguistics* (2011).

[2]We attempted to fill in this gap by running some of the code that was in the code base when we started, but it was projected to take 10 days to complete. Moreover, the program encountered an `OutOfMemoryError` exception.

[3]We tried to use both `QNMinimizer` and `RISO`.